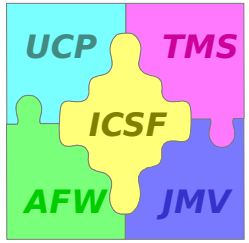
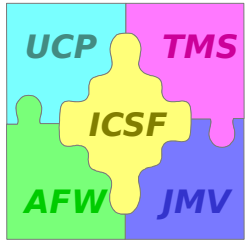


Software Developer's Guide (SDG)

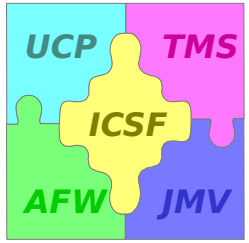
Peter Kunkel

LOGICON INRI

S/W Development Goals

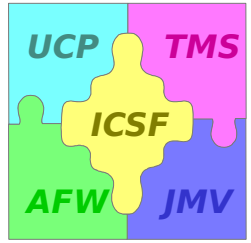


- Install the needed SDKs within your software develop environment.
- Use the SDKs including setting up the environment, i the compiler(s), and compiling the example source co
- Build software applications that use the services prov ICSF SDKs.
- Test software applications in your software developm environment without having to install the entire DII C runtime environment.



It is up to each developer to fully understand his or her own development environment

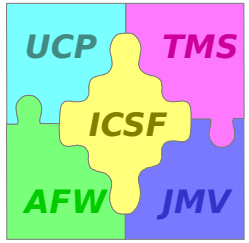
Software Development Kits (SDKs)



SDK Name	Required SDKs
IFL	None
JMV	IFL
AFW	IFL, JMV
UCP	IFL
TMS	IFL, UCP
TMSV	IFL, AFW,
TMS JMVMD	JMV

- UCP is required by TMS only if you want to send in track data via a co channel and/or receive messages to be decoded.
- TMSV and JMVMD are not SDKs but support other SDK segments.
- TMSV required only if you want to display a chart with track.

ICSF SDK Directories



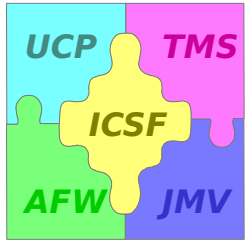
app defaults	A variety of application default files used by th applications and libraries.
bin	Binary executable files packaged with the SDK
class	A collection of Java classes.
data	A minimum set of data files used by the SDK.
include	Include files required to use the API functions.
lib	Library files, static and dynamic, required for the API functions.
utils	Additional utilities that can be used to demons general SDK functionality.
Scripts	Collection of Installation/Environment shell script

SDK Environment Variables

Environment Variable	UNIX Script (IFLSDK_ENV.csh) *	WindowsNT Script (IFLSDK_ENV.bat) *
ICSF_HOME	must be set	must be set
JAVA_HOME	must be set	must be set
JAVA_PROG	must be set	must be set
DATA_DIR	$ICSF_HOME/data	%ICSF_HOME%/data
REGISTRY	$ICSF_HOME/Registry/Registry	%ICSF_HOME%/Registry/Registry

* Each SDK has its own environment script which sets up the environment for that SDK.

Use of Environment Variables



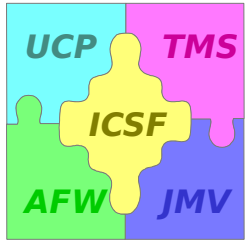
ICSF_HOME: This variable should be set to the top-level directory SDK is located.

DATA_DIR: This variable should be set relative to the ICSF_HOME and should indicate the location of the "Data" directory, the c repository of data shared among the ICSF SDKs.

REGISTRY: This variable should be set relative to the ICSF_HOME and should indicate the location of the "Registry" directory, t repository for configuration items shared among the ICSF SI

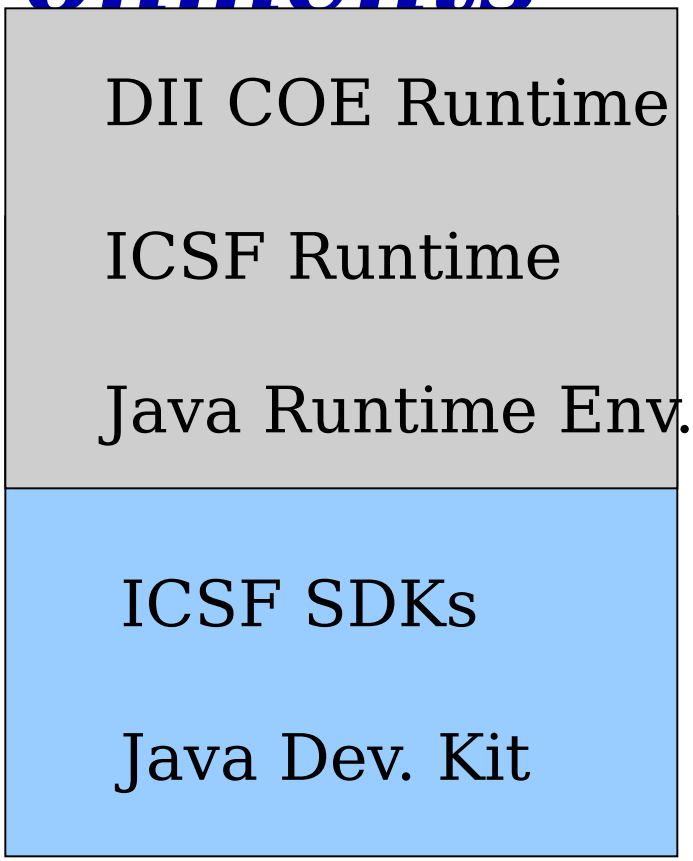
JAVA_HOME: This variable should point to the location of the devel if one is installed. This variable is used toextend the user's PA CLASSPATH variables.

JAVA_PROG: This variable should designates the executable to be r running Java classes (i.e., java).

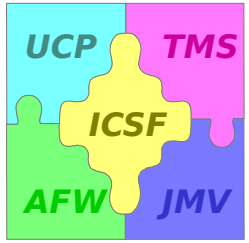


The ICSF SDK environments build on one another. For e
if you run the IFLSDK_ENV.csh script, which sets up sp
development environment variables, and then move into
xterm to install and use another ICSF SDK, JMV for exa
will need to source the IFLSDK_Inst.csh script again (in
xterm) before you can install and use the JMV SDK.

Choice of Development Environments

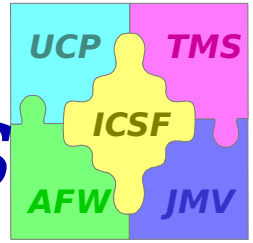


Installing ICSF SDKs




```
setenv ICSF_HOME /h/ICSF
setenv JAVA_HOME /usr/java
setenv JAVA_PROG java
setenv TMSV_HOME $ICSF_HOME/TMSV
setenv REGISTRY $ICSF_HOME/data/registry/registry

source $ICSF_HOME/IFLSDK/Scripts/IFLSDK_ENV.csh
$ICSF_HOME/IFLSDK/Scripts/IFLSDK_Inst.csh

source $ICSF_HOME/JMVSDK/Scripts/JMVSDK_ENV.csh
$ICSF_HOME/JMVSDK/Scripts/JMVSDK_Inst.csh

source $ICSF_HOME/AFWSDK/Scripts/AFWSDK_ENV.csh
$ICSF_HOME/AFWSDK/Scripts/AFWSDK_Inst.csh

source $ICSF_HOME/UCPSDK/Scripts/UCPSDK_ENV.csh
$ICSF_HOME/UCPSDK/Scripts/UCPSDK_Inst.csh
```

Installing ICSF SDKs (2)



source $ICSF_HOME/TMSSDK/Scripts/TMSSDK_ENV.csh
$ICSF_HOME/TMSSDK/Scripts/TMSSDK_Inst.csh

source
$ICSF_HOME/TMSVSDK/Scripts/TMSVSDK_ENV.csh
$ICSF_HOME/TMSVSDK/Scripts/TMSVSDK_Inst.csh

source
$ICSF_HOME/JMVMDSDK/Scripts/JMVMDSDK_ENV.csh
$ICSF_HOME/JMVMDSDK/Scripts/JMVMDSDK_Inst.csh

Group and Password Modifications

In order to properly activate the runtime processes for each SDK on the UNIX machines, it is necessary to add several groups to the ***/etc/group*** file and to add the developer's "user" to the ***/etc/passw***

1. In the /etc/group file, add the following groups:

   ICSF::600:<user name>
   TMS::666:<user name>,ICSF
   TMSElint::667:<user name>,ICSF
   UCP::500: <user name>,ICSF

2. In the /etc/passwd file, add the following user:

   TMS:x:666:666::<home directory of the TMSSDK>:/bin/
   UCP:x:500:500::<home directory of the UCPSDK>:/bin/c
   ICSF:x:600:600::<home directory of the IFLSDK>:/bin/c

/etc/service Additions

In order to properly activate the runtime processes for each SDK on the UNIX machines, it is necessary to add several services to the ***/etc/services*** file.

```
Importer            2095/tcp
ChartIII            2307/tcp
UnitService         7893/tcp
UCPServ             2030/tcp
UCPMpr              2035/tcp
UCPNetwork          2020/tcp
#UCPChnls           2100-2200/tcp
tdbm                2000/tcp
tdbm                2000/udp
tmsevtlstnr         2001/tcp
gdbm.idb            2002/tcp
gdbm.idb            2002/udp
bcst                2050/tcp
DFLRPC              2040/tcp
DFLRPC              2040/udp
track-man           2025/tcp
```

SDK Environment Script

- Check for ICSF_HOME, JAVA_HOME, and JAVA_PROG.

- Verify that the environment variables needed by the SDK are set.

- Set the <SDK>_HOME environment variable:

  setenv IFL_HOME $ICSF_HOME/ IFLSDK

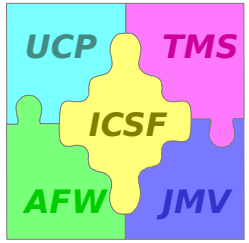
- Add the SDKs class directories to the CLASSPATH.

  setenv CLASSPATH "{CLASSPATH}:${IFL_HOME}/class”

- Add <SDK>_HOME/lib to the shared library path.

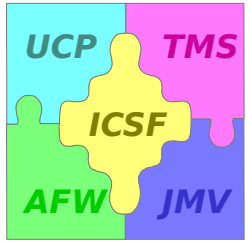
  setenv LD_LIBRARY_PATH "{LD_LIBRARY_PATH}:${IFL_HOME}/lib”

Establishing the Development Environment




```
setenv ICSF_HOME /h/ICSF
setenv JAVA_HOME /h/JAVA2
setenv JAVA_PROG java
setenv TMSV_HOME $ICSF_HOME/TMSV
setenv REGISTRY $ICSF_HOME/data/registry/registry

source
$ICSF_HOME/IFLSDK/Scripts/IFLSDK_ENV.csh
source
$ICSF_HOME/JMVSDK/Scripts/JMVSDK_ENV.csh
source
$ICSF_HOME/AFWSDK/Scripts/AFWSDK_ENV.csh
source
$ICSF_HOME/UCPSDK/Scripts/UCPSDK_ENV.csh
source
$ICSF_HOME/TMSSDK/Scripts/TMSSDK_ENV.csh
source
```

Running ICSF Servers



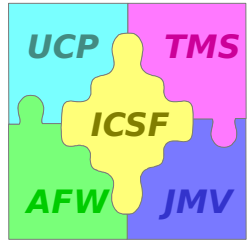
Setup ICSF Development Environment

IFLRun -session &
JMVRun &
UCPRun -um 007 &
TMSRun -boot &

Wait about 60 seconds

AFWRun

IFL Overview



Integrated library of basic program subcomponents that supports:

- **Joint Mapping Tool Kit - Visualization (JMV),**
- **Tactical Management System (TMS),**
- **Universal Comms Processor (UCP), and**
- **C4I Common Extensions (COMEXT).**

Software application developers can use IFL routines in writing, building, and running applications in two ways:

routines called from IFL (used directly by the programmer)

- **routines called indirectly through calling the code already**

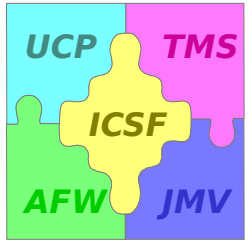
Development in "C"

- VHelp() has been deleted in IFLSDK (since Version 1.2) in favor of DII COE-based Help.

- ***All other IFLSDK public functions are "obsolesced,"*** which means that the functions are supported in IFLSDK 4.2.0.0 but may not be supported beyond this build.

> Note:
> All subscriber programs should transition to alternative functions. Recommendations are provided in *Recommended Replacement Functions* of the IFLSDK.

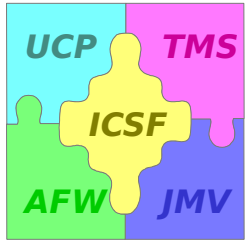
Development in Java



IflToolBar - Describes the toolbar functionality provided

IflDockingViewPanel - Describes how an application ca
integrate into the chart window.

IflToolBar



IflToolBar was conceived to fill a gap in service that the ***JToolBar*** did not cover, which was to have more then on toolbar per container.

If someone were to use more then one ***JToolBar*** in a giv container, the effects could be less then desired -- rangi from basic inability to dock toolbars in appropriate area the loss of toolbars.

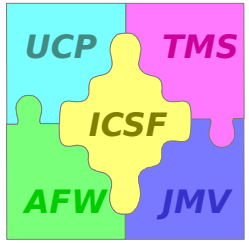
Creation of an IflToolBar

- Each toolbar must know which container is responsible for its management. This is the reason for the mandatory Jcomponent parameter in each ***IflToolBar*** constructor.

- This constructor parameter can be retrieved from the ***IflFrame***.

1) JComponent panel = myIflFrame.getCenterPane();
2) IflToolBar ifltb = new IflToolBar ("Sample", panel.getCenterPane());

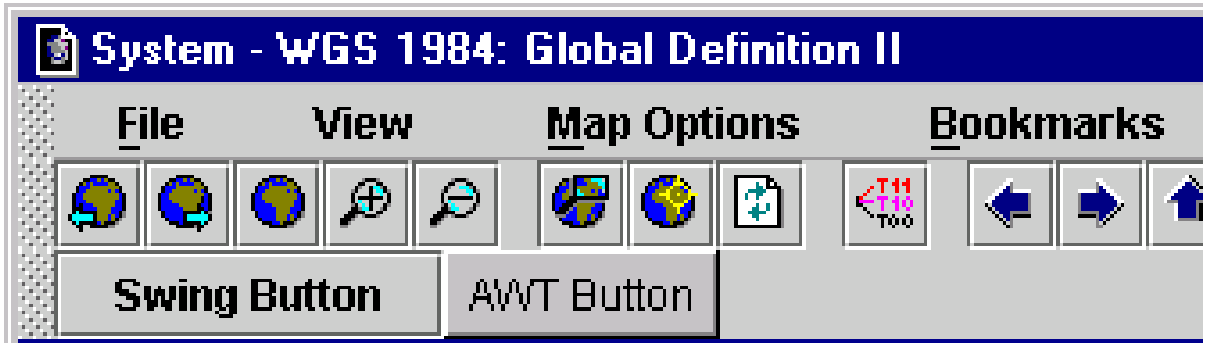
Adding Components to Toolbar



- Now that the toolbar has been created, it is ready to a components. The toolbar can hold any number of components and should work like any other Java / S container. There is no restriction on the types of c that can be placed inside a toolbar, so either native AWT or Swing objects can be managed.

```
3) ifltb.add(new JButton("Swing Button"));
4) ifltb.add(new Button("AWT Button"));
```

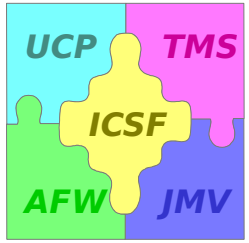
Managing the Toolbar



5) panel.add(ifltb, IflToolBarLayout.NORTH);

In line 5 we added our toolbar to the center pane of th
IflFrame, which uses an IflToolBarLayout manager.

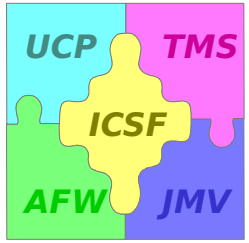
IflDockingViewPanel



IflDockingViewPanel was created to help programmer present a logical view in which end users can interact. It allows the end user to:

- Resize the component view.
- Access a menu system for the component view.
- Close the component view on demand.
- Move the view to a desired location.

IflDockingViewPanel Setup



Before an **IflDockingViewPanel** can be created some in
setup is required. Most of these setup items are optional
first item to set up is a docking view **closing listener**. T
allow the window closing behavior in the IflDockingView
to be controlled.

```
1) private class ClosingListener
2)     implements IflViewAdapter{
3)
4)     public void viewClosing(IflViewEvent ive){
5)         if(!dataValidation())
6)             ive.consume();
7)     }
8) }
```

Adding Docking View Menu

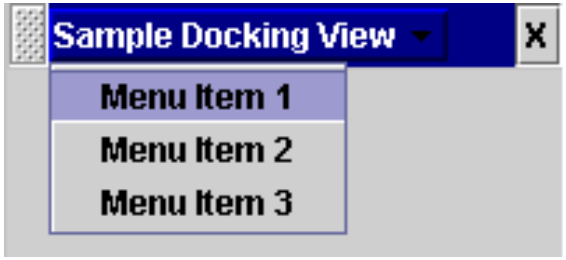
To make menu usage easier, ***JPopupMenu*** is used. The JPopupMenu is a common Java Swing item.

```
 9) JPopupMenu myPopupMenu = … creation of Menu …;
10) ClosingListener myClosingListener = new ClosingListener();
11) JComponent centerPanel = myFrame.getCenterPanel();
```

Lines 9 and 10 are just setup variables so they can be referenced when it comes time to actually create the docking view.

Line 11 is needed because this is a sub-class of IflToolBar and we need that center panel for the layout manager to perform docking view synergy.

Creation of IflDockingViewPanel



To create the docking view panel:

```
12) IflDockingViewPanel idvp = new IflDockingViewPanel("Sample Docking View",
13)                                 IflFrameCenterPanel,
14)                                 myClosingListener,
15)                                 myPopupMenu);
```

Lines 12-13 are required. They give the docking view a title and provide infoto the docking view on the managing component.

Line 14: If you don't want a docking view close button leave it null